Decentralized Fault Detection and Management for Wireless Sensor Networks

Ka Lok Man[1], Chen Chen[2], Danny Hughes[1]

[1.] Xi'an Jiaotong-Liverpool University, Suzhou Industrial Park, Suzhou, China
{ ka.man, daniel.hughes }@xjtlu.edu.cn

[2.] Global Institute of Software Technology, Suzhou, China
catherinechenchen@gmail.com

***Abstract*— Wireless Sensor Networks are increasingly being deployed in long-lived, challenging application scenarios which demand a high level of availability and reliability. To achieve these characteristics in inherently unreliable and resource constrained sensor network environments, fault tolerance is required. This paper presents a generic and efficient fault tolerance algorithm for Wireless Sensor Networks. In contrast to existing approaches, the algorithm presented in this paper is entirely decentralized and can thus be used to support fully autonomic fault tolerance in sensor network environments.**

Keywords: Wireless Sensor Networks, Fault Tolerance.

I. INTRODUCTION

A Wireless Sensor Network (WSN) is composed of tiny embedded computers equipped with sensors and low-power radios known as 'motes'. These motes form self-organizing networks that are capable of sensing and reacting to the physical environment [1]. WSNs have been deployed to tackle many critical real-world problems such as habitat monitoring [2], flood warning [3] and fire detection [4]. Unfortunately, while WSNs hold great promise for a number of application domains, they remain unreliable due to a number of technical, environmental and application-centric challenges.

In terms of *technical challenges*, the low-power wireless networking protocols used in WSNs are inherently unreliable, being subject to packet loss, network segmentation and node mobility. Furthermore, as mote platforms are very resource constrained, there is limited capacity available to host fault tolerance functionality.

In terms of *environmental challenges*, WSNs are often deployed in hostile environments where motes may be damaged or destroyed: for example, consider the flood warning scenario described in [3], or the fire detection scenario described in [4]. Along with node failure, WSNs networking may be degraded by the environmental factors such as radio interference and adverse weather conditions.

In terms of *application-centric challenges*, the majority of WSNs applications are expected to operate at large scale without human intervention for long periods of time. Furthermore, WSNs are increasingly being viewed as multi-application infrastructures that are expected to host diverse applications [5] that may be safety-critical [3, 4].

Addressing the technical, environmental and application-centric challenges highlighted above necessitates the deployment of fault detection and management functionality. To be successful, any fault detection and management approach for WSNs must demonstrate the following properties:

- *Efficiency:* given the scarce computational resources available on motes such as the T-mote Sky [6], Mica-Z [7] and Sentilla Perk [8], any fault tolerance approach must be very efficient and light-weight.
- *Scalability:* the large scale of sensor networks coupled with the limited bandwidth of WSN network technologies necessitates that any fault detection and management approach must be extremely scalable, imposing minimal message passing overhead.
- *Autonomic:* as WSNs are often deployed in remote locations and are expected to operate unattended for long periods of time, a successful approach to fault detection and management should be autonomic and decentralized.
- *Application Independence:* while initial sensor network deployments such as [1], were tightly coupled with their host target application, current trends in sensor network architectures are towards a multi-application, shared infrastructure paradigm [5]. This necessitates a decoupling of fault tolerance functionality from application logic.

This paper introduces the Self Fault Management (SFM) architecture for WSNs. In contrast to much of the existing work in this area [9, 10, 22], SFM is lightweight, decentralized and provides a generic base upon which to build application-independent fault tolerance functionality. Critically, simulation shows that SFM scales well, and is thus capable of providing efficient fault tolerance in wireless sensor networks.

The remainder of this paper is structured as follows: Section II discusses related work. Section III presents the design of a decentralized fault tolerance scheme for WSNs. Section IV presents the proposed architecture for fault tolerance. Section V evaluates the proposed approach. Section VI highlights directions for future work. Finally, concluding remarks are given in section VII.

978-1-4244-6949-9/10/$26.00 ©2010 IEEE

II. RELATED WORK

Fault tolerance is a critical component of any WSN application. In 'Fault Tolerance in Wireless Sensor Networks' [12], Koushanfar et al. review the suitability of classic distributed systems fault tolerance techniques for WSNs. At the *physical layer*, *system software*, and *middleware levels*. Fault tolerance techniques at each of these three layers are reviewed in Sections A to C below.

A. Fault Tolerance at the Physical Layer

At the *physical layer*, a range of adaptive link-layer techniques have been applied to improve the reliability of wireless networks including [13] and [14]. These techniques have now largely been incorporated into WSN technologies at the physical layer 802.15.4 [15] and MAC layer. For example TSMP [16] uses a frequency hopping approach to minimize packet loss due to radio interference. While this category of fault tolerance approaches may reduce network faults and thus improve network performance, it is not possible for these approaches to eliminate faults entirely, which may also arise due to nodes moving out of range, crashing or even being destroyed. This necessitates fault tolerance functionality at the higher software layers.

B. Fault Tolerance at the System Software Layer

At the *system software layer*, early approaches focused on sensor calibration [17, 18], and has made significant strides in ensuring the accuracy of low-level sensor readings. More recently Sundaram et al. [19] integrated lightweight tracing functionality with the lightweight TinyOS WSN operating system [1], in order to allow for the distributed tracing and testing of WSN software. While these approaches may be used to ensure the correctness of sensor readings and software respectively, they provide no support for avoiding dynamic faults such as those caused by node failure.

C. Fault Tolerance at the Middleware Layer

At the *middleware layer*, approaches to ensuring fault tolerance in WSN have focused primarily on adaptive duty cycling which has been used to minimize faults due to battery depletion, as described in [21]. When coupled with redundant deployment of WSN motes, adaptive duty cycling can also be used to provide enhanced fault tolerance as described in [20]. Most recently, de Jong et al. introduced MoMi, [11] a model driven middleware approach to identifying and diagnosing node failure and disconnection faults in WSNs. While MOMi provides good support for the centralized detection of faults in WSNs, it does not provide support for the decentralized detection of faults, making it unsuitable for many network architectures.

D. SFM in the Context of Related Work

The work presented in this paper may be viewed as a fault detection layer that operates at the system software level, exposing generic fault detection to the higher middleware layers. As such, our work is inherently complimentary to fault tolerance approaches that operate at the physical layer [13, 14], and those approaches which focus on proving the correctness of the sensing system [17] [19]. Furthermore, the generic functionality provided in our approach could easily be used to support higher level middleware such as MOMi [11].

III. DESIGN OF SFM

As discussed in the previous section, SFM focuses on detecting faults that may occur at the *physical* and *system software* layers of WSN applications. Section II.A examines the sources of faults at these levels of the WSNs. Section II.B then provides a list of concrete requirements against which SFM will be evaluated.

A. Sources of Faults in WSNs

Faults at the physical and systems software layers of WSNs may arise from a four primary sources, each of which are reviewed below:

- *Node / link failure*: WSN motes may become inaccessible due to the failure of mote hardware (e.g. due to battery depletion), or failure of networking facilities (e.g. due to occlusion). In either case, this kind of fault may be minimized through the use of adaptive link layer technologies as described in Section II.A and detected by the inaccessibility of a mote.
- *Node movement*: In mobile scenarios, motes may move out of range of each other, causing link failure as described above, which may be detected by the inaccessibility of a mote. This kind of fault may be minimized through the use of mobility aware protocols such as MAODV [22] and may be detected by the inaccessibility of a mote.
- *Software crashes*: Software crashes may occur due to software bugs, which can be particularly hard to detect in WSNs, as discussed in [23]. In this paper, we focus on those bugs which render the mote inoperable and thus may be detected by the inaccessibility of a mote. For bugs with more complex systems, our approach may be paired with tracing and bug identification software such as [19] or [23].

B. Requirements

The context gained through examination of related work in Section II and the analysis of WSN faults provided in Section II.A, enables the enumeration of requirements for SFM:

- *Detection of non-responsiveness*: as discussed in Section II.B, non-responsiveness can be used to detect faults due to *node / link failure*, *node movement* and *software crashes*.
- *Detection of inaccurate sensor readings*: detection of inaccurate sensor readings may be accomplished by comparing current readings to recent observations and detecting significant deviations.
- *Platform independence*: SFM should be sufficiently simple and portable that it could be easily realized on diverse WSN runtimes including NesC-based systems such as TinyOS [1], C-based systems such as Contiki [24] and Java-based systems such as SQUAWK [25].

- *Decentralized architecture*: in order to provide support for diverse and potentially dynamic network architectures, the SFM fault detection system should be decentralized.
- *Low network overhead*: given the resource constrained nature of current WSN platforms, it is essential that SFM imposes minimal computational and network overhead.
- *Timely detection of faults*: while it is essential that the system imposes minimal overhead, it is equally important that the system is able to reliably diagnose faults in a timely fashion.

Section IV describes the architecture that was realized to meet these requirements. This architecture is then evaluated in Section V.

IV. SELF FAULT MANAGEMENT ARCHITETCURE

SFM assumes a decentralized network architecture, which is composed entirely of standard motes with no special function devices. Each node participates in self-fault management and nodes in the same neighborhood collaborate to realize self-fault management functions. We consider a set of N sensor nodes, wherein every sensor node has a unique and irreversible ID and two possible states: *FAULTY* and *NON-FAULTY*. As discussed in Section III, we only consider faults that may be detected through the non-responsiveness of a mote. Once a node is *FAULTY*, it will no longer be used to provide the information and instead will fill the role of a mesh router. In our initial work, we make three simplifying assumptions. Firstly, we assume that the nodes in the WSN are static or have only limited mobility that will not result in topology change. Secondly, we assume that no new nodes will be added to the network during the interval between fault detection and fault recovery. Thirdly, in this paper, we will not consider the duty-cycling aspects of WSN management.

In distributed self-fault management each node uses a set of counters, which are incremented when fault-like behavior is exhibited by any of their neighbors. All the counters are periodically reset to zero. This architecture uses a table (*ctMap*) that is used to store these counters along with the VOTE message type for common nodes collaborating to diagnose suspicious node. The main parameters used in SFM messages are shown in Listing 1. Listing 2 shows the data structures used to store fault detection information.

Listing 1 - Parameters Used in SFM Messages

```
originator  (message creators ID),
src         (last sender's ID),
ttl         (broadcast range),
sv          (sensor value),
susID       (suspicious node ID),
isSus       (VOTE - true indicates faulty),
malID       (faulty node ID).
```

Listing 2 – Data Structures for Storing Information:

```
ssvQue      (latest sensor values)
malVec      (used to store faulty node IDs)
ctMap       (per-neighbor fault counters)
```

A. Suspect Tagging Process

Each node periodically broadcasts a HELLO message to its neighbors containing the parameters *originator* and *sv* which describe the node from which the message originates and that nodes current sensor value. When a node receives a HELLO message, if the originator of the message is not in *malVec* (the faulty node data structure), a comparison between *sv* (the current sensor value) and *sasv* (the node's current average sensing value) is made. If the difference between these values exceeds a predefined threshold: θ, the corresponding fault counter will increase. When the counter exceeds a certain threshold η, the node will become the *initiator* and broadcast a SUSPECT message. At least three neighbors are required to diagnose a fault and therefore, if a one-hop broadcast does not reach sufficient motes, a two-hop broadcast will be used and so on, until enough nodes are contacted. This process is illustrated in Figure 1 below:

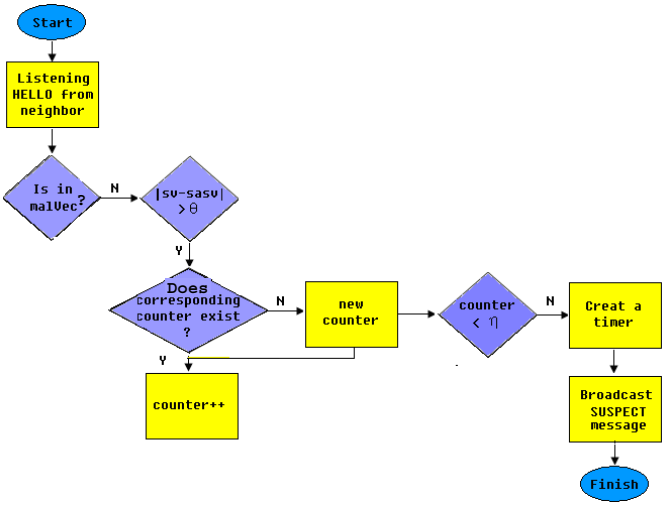


Figure 1 – Suspect Tagging Process

B. Self-Fault Determination

When a node receives a SUSPECT message, the node first checks whether *susID* is already in *malVec*. If *susID* already exists in *malVec*, a DETERMINE message will be sent back identifying the node as faulty. Otherwise, the node will check whether *susID* is in their neighbor table. If the suspicious node is one of its neighbors, it will send back a VOTE message with its vote result (*isSus*). Only when the corresponding counter value reaches a certain threshold μ, the parameter isSus will be set to true. This process is shown in Figure 2.

The initiator node acts as the arbiter for determining faults. As mentioned previously, when a SUSPECT message is generated, a timer is also initiated. When this timer expires, the initiator node checks the counter *susCounter* to determine the number of agree VOTE messages (where *isSus* = true). If the value of susCounter is larger than a predefined value φ, this initiator will three-hop broadcast a DETERMINE message as shown in Figure 2.

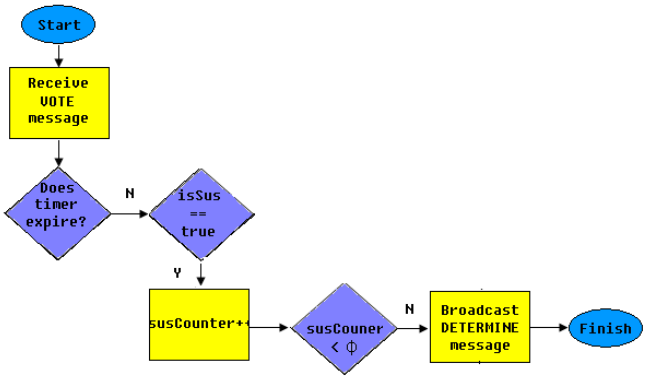


Figure 2 – Self-Fault Determination

C. *Self-Fault Recovery*

The recovery phase in distributed self-fault management architecture occurs as follows: when a node receives a DETERMINE message, it first checks whether the faulty node in its neighbor table (NT) and whether the ID of faulty node is already in the *malVec*. If the faulty node is its neighbor and has not been isolated, it will disconnect the link to faulty neighbor. Figure 3 shows the faulty node isolation process.

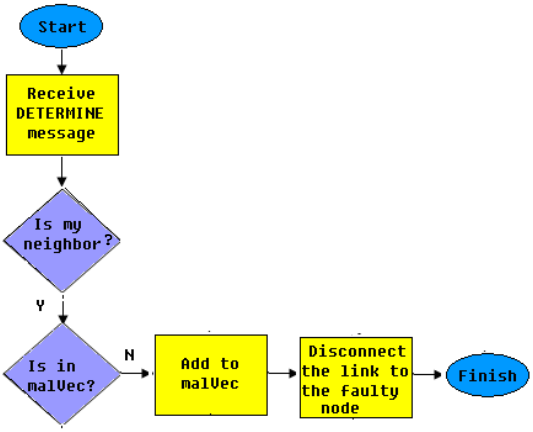


Figure 3 – Self-Fault Recovery

V. EVALUATION

In this section we will evaluate the performance of the self fault management architecture through simulation. Section V.A describes our simulation environment. Section V.B then reports on the results of this simulation.

A. *Simulation Environment*

We used the Objective Modular Network Testbed in C++ (OMNET++) as our simulation platform [26]. OMNET++ is an object-oriented, modular discrete event simulation system. This simulator can be used for wired and wireless network communication modeling, network protocol modeling and many other discrete event approaches. An OMNET++ model is formed by modules, messages, gates and links. Some simple modules can be nested or grouped into a compound module. The communication between modules is by message exchanging. We use version 4.0 of OMNET++, which is Eclipse based [26]. In our simulation, each mote, or node contains four sub-modules. The sizes of the network models are range from 16 nodes to 128 nodes and the number of faulty nodes range from 1 to 8.

- *Physical Model*: The physical model represents a node's physical layer, including the neighbors of the node, and is responsible for communication with other nodes as well as the fault recovery process realized on this model. When a node receives a message from a remote source, the message will first arrive on physical layer and then be passed to the upper layers.
- *MAC Model*: the MAC model realizes the functions for Media Access Control. There is a buffer to control the message received from other nodes. If the node is busy, the message will be stored into the buffer. When the buffer is full, the received message will be deleted. So if there are a lot of traffic, the message loss rate will increase, that will impact the performance of fault management and cause delay.
- *Route Model*: the route model is where routing protocols are implemented. In our simulation, we use AODV, which is a commonly used protocol in WSNs [22]. However, AODV has one major disadvantage: when the routing table is empty, it takes a long time to fill.
- *App Model*: this corresponds to the application layer in the communication stack. Our self-fault management is implemented on this layer. The fault management messages are generated here and send through route, MAC, and physical to other nodes. When a fault is diagnosed, a HEAL message will be sent directly from application layer to physical layer in order to disconnect the link to the faulty node.

B. *Network Overhead*

Traffic overhead will lead to the higher energy consumption, which in turn leads to shorter battery life. Therefore, a good self-fault management for WSNs should minimize energy consumption due to message passing.

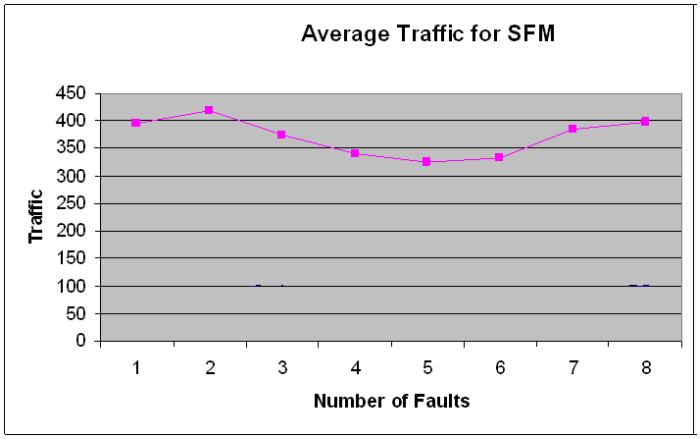
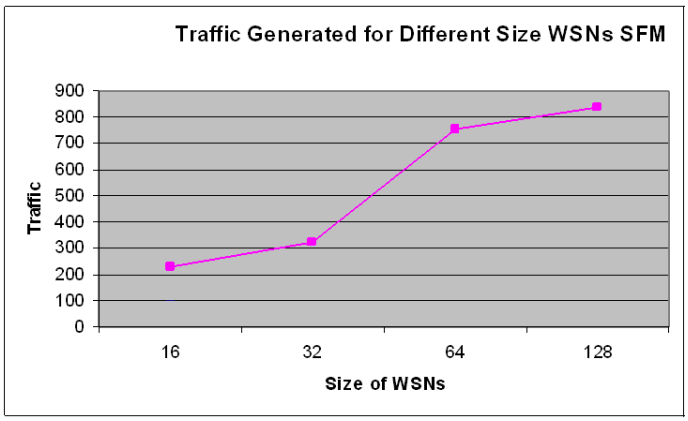


Figure 4 – Traffic Generation with Number of Faults

Figure 4 shows eight networks, which all contain 128 sensor nodes, but the number of faulty nodes range from one to eight. The average number of messages exchanged from fault

diagnosis to recovery is around 350, which is quite reasonable for a network of this size.

Figure 5 illustrates four sizes of WSNs which respectively contain 16, 32, 64 and 128 nodes where two nodes will become faulty. In order to detect, diagnose and recovery, the traffic overhead for distributed management increases with the size of the WSNs. This is because the number of neighbours of each node increases with WSN size.



. **Figure 5 – Traffic Generation with WSN Size**

C. *Fault Detection Time*

Time cost indicates the time required to diagnose and recover from a fault. Some applications require fast recovery from faults and in general, the faster a fault is recovered from, the lower the problems that will be caused.

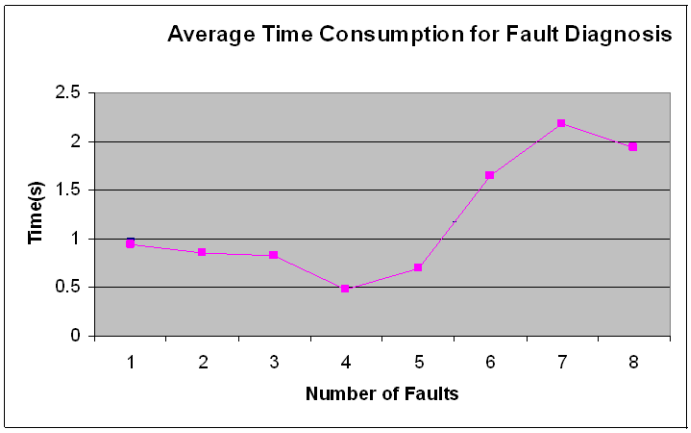


Figure 6 – Time Required to Diagnose Faults

Figure 6 shows the average time required for fault diagnosis in centralized and distributed simulation, which is the time interval from when the first or initial node sent out the SUSPECT message to the time when the initial node send out the DETERMINE message. In distributed simulation, because the neighbors of faulty nodes cooperate to provide fault management function, so the time consumption does not have big difference when the number of faulty nodes increases.

D. *Success Rate*

Success rate denotes the *quality* of self-fault management. A well quality fault managed WSN which needs to have the capacity of high success rate to diagnose and recover a fault.

Figure 7 compares eight networks which all contain 128 nodes but have different number of faulty nodes. As the number of fault nodes increases, the probability for diagnosis failure also increases. The main reason for this is message loss. However, SFM still successfully isolates all faults in our simulation.

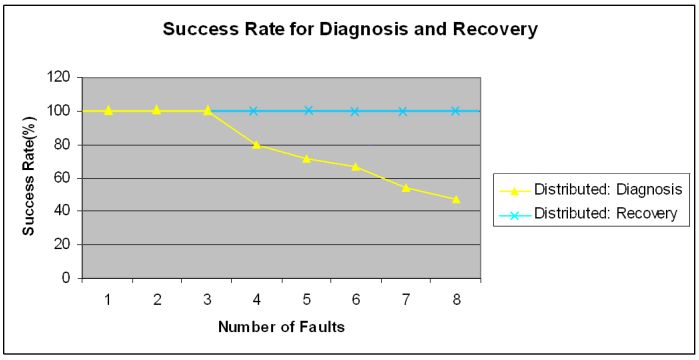


Figure 7 - Success Rate for Fault Diagnosis

VI. FUTURE WORK

In our future work, we intend to take better account of network dynamism, including mote duty-cycling (i.e. mote wake and sleep cycles), networks with high rates of churn (i.e. where motes frequently join and leave the network) and networks with high rates of mobility. The latter presents a particular challenge for the work presented in this paper as faults may be transient and the network will naturally recover.

VII. CONCLUSIONS

This paper has presented the Self Fault Management (SFM) framework for WSN. SFM is designed to provide decentralized and application independent fault detection. SFM represents a concrete addition to the state-of-the-art in WSN fault detection and management and may be used in a complementary fashion with lower level link fault tolerance approaches [13], [14] and also to support higher level fault diagnosis middleware such as [11]. Evaluation has shown that SFM achieves these design goals while offering highly reliable fault detection, with minimal network overhead.

ACKNOWLEDGEMENTS

Ka Lok Man and Danny Hughes would like to thank Xi'an Jiaotong-Liverpool University, China for supporting and funding the dissemination of the work presented in this paper.

REFERENCES

[1] Hill J., Szewczyk R., Woo A., Hollar S., Culler D., Pister K., System Architecture Directions for Networked Sensors, in ACM SIGPLAN, Vol. 35, No. 11, November 2000, pp. 93-104.

[2] Mainwaring A., Polastre J., Szewczyk R., Anderson J., Wireless Sensor Networks for Habitat Monitoring, in Proc. of 1st ACM

International Workshop on Wireless Sensor Networks and Applications, Atlanta, Georgia, USA, 2002, pp 88 – 97.

[3] Hughes D., Greenwood P., Coulson G., Blair G., Pappenberger F., Smith P., Beven K., An Experiment with Reflective Middleware to Support Grid-based Flood Monitoring, in Wiley Inter-Science Journal on Concurrency and Computation: Practice and Experience, vol. 20, no 11, November 2007, pp 1303-1316.

[4] Coulson G., Gold R., Lad M., Mascolo C., Mottola L., Picco G.P., Zachariadis S., Dynamic Reconfiguration in the RUNES Middleware, in Proc. of the 3rd IEEE International Conference on Mobile Ad-hoc and Sensor Systems, Vancouver, Canada, 2006, pp. 574-577.

[5] Hughes D., Thoelen K., Horré W., Matthys N., Michiels S., Huygens C., Joosen W., LooCI: A Loosely-coupled Component Infrastructure for Networked Embedded Systems, in the proceedings of the 7th International Conference on Advances in Mobile Computing & Multimedia (MoMM'09), 2009.

[6] MoteIV, T-MOTE Sky Ultra-low Power Wireless Module Data Sheet: http://www.cs.uvm.edu/~crobinso/mote/tmote-sky-datasheet-102.pdf.

[7] Crossbow, MICA-Z Wireless Measurement System, Data Sheet: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf.

[8] Sentilla, Perk Platform Frequently Asked Questions: http://www.sentilla.com/perk_faq.html.

[9] Koushanfar F., Potkonjak M., Sangiovanni-Vincentelli A., Fault Tolerance in Wireless Ad Hoc Sensor Networks, in IEEE Sensors, Vol. 2, 2002, pp. 1491-1496.

[10] Sundaram V., Eugster P., Zhang X., Lightweight Tracing for Wireless Sensor Networks Debugging., in the Proc. of the 4th International Workshop of Middleware for Wireless Sensor Networks (MidSens'09), Urbana Champaign, Illinois, USA, 2009.

[11] de Jong A., Woehrle M.Langendoen K., MoMi – Model-Based Diagnosis Middleware for Sensor Networks in the Proc. of the 4th International Workshop of Middleware for Wireless Sensor Networks (MidSens'09), Urbana Champaign, Illinois, USA, 2009.

[12] Koushanfar F., Potkonjak M., Sangiovanni-Vincentelli A., Fault Tolerance in Wireless Sensor Networks, chapter in the Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems, CRC Press, 2004.

[13] Eckhardt D. A., Steenkiste P., Improving Wireless LAN Performance via Adaptive Local Error Control, in proc. of 6th International Conference on Network Protocols, ICNP'98, Austin, Texas, USA, 1998.

[14] Eckhardt D.A., Steenkiste P., A Trace-Based Evaluation of Adaptive Error Correction for a Wireless Local Area Network, in Journal on Special Topics in Mobile Networking and Applications (MONET), Issue 4, 1999, pp. 273–287.

[15] 802.15.4 Working Group, available online at: http://www.ieee802.org/15/pub/TG4.html

[16] Pister K. S. J., Doherty L., TSMP: Time Synchronized Mesh Protocol, in proc. of the International Symposium on Distributed Sensor Networks (DSN08), November 2008, Orlando, Florida, USA.

[17] Whitehouse K., Culler D., Calibration as Parameter Estimation in Sensor Networks, in proc. of 1st International Workshop on Wireless Sensor Networks and Applications, WSNsA'02, Atlanta, Georgia, USA, September 2002

[18] Bychkovskiy V., Megerian S., Estrin D., Potkonjak M., Colibration: A Collaborative Approach to In-Place Sensor Calibration." 2nd International Workshop on Information Processing in Sensor Networks (IPSN'03), April 2003, pp. 301-316, April 2003.

[19] Sundaram V., Eugster P., Zhang X., Lightweight Tracing for Wireless Sensor Networks Debugging, in proc. of 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks (MidSens'09), Urbana Champaign, Illinois, USA, December 2009, pp. 13-18.

[20] Yuan T., Zhang S., Secure Fault Tolerance in Wireless Sensor Networks, in proc. of 8th International Conference on Computer and Information Technology Workshops (CIT-Workshops'08), Sydney, Australia, 2008, pp. 477-482.

[21] Speer A.P., Chen, I. R., Effect of redundancy on the mean time to failure of wireless sensor networks, in Concurrency and Computation: Practice & Experience, Vol. 19, Issue 8, pp. 1119 – 1128

[22] Royer E., Perkins C., Multicast ad hoc on- demand distance vector (MAODV), IETF draft, available online at: http://tools.ietf.org/html/draft-ietf-manet-maodv-00

[23] Sasnauskas R., Bitsch J. A., Alizai M. H., Wehrle K.,KleeNet: automatic bug hunting in sensor network applications, in proc. of 6th ACM conference on Embedded network sensor systems (Sensys'08), Raliegh, North Carolina, USA, pp 425-426.

[24] Dunkels A., Grönvall B., Voigt T., Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors, in proc. of the 29th International Conference on Local Computer Networks (LCN'04), Tampa, Florida, USA, 2004, pp. 455-462.

[25] Simon D., Cifuentes C., Cleal D., Daniels J., White D., Java on the Bare Metal of Wireless Sensor Devices: the Squawk Java Virtual Machine, in Proc. of the 2nd International Conference on Virtual Execution Environments, Ottawa, Canada, June 2006, pp 78-88.

[26] OMNET++ Simulation Environment - User Community, available online at : http://www.omnetpp.org/.